Triggers Part - I

- Jayendra Khatod

Objectives

- Describe different types of triggers
- Describe database triggers and their use
- Create database triggers
- Describe database trigger firing rules
- Remove database triggers
- Understand the rules governing triggers
- Implement triggers

Overview of Triggers

- **A trigger is a PL/SQL block or a PL/SQL procedure associated with a table, view, schema, or the database**
  - **A trigger is a PL/SQL block that executes implicitly whenever a particular event takes place.**
  - **A trigger can be either a database trigger or an application trigger.**

Types of Triggers

A trigger can be either:

- **Application trigger: Fires whenever an event occurs with a particular application**
- **Database trigger: Fires whenever a data event (such as DML) or system event (such as logon or shutdown) occurs on a schema or database**

Designing Triggers: Guidelines

- **Design triggers to:**
  - **Perform related actions**
  - **Centralize global operations**
- **Do not design triggers:**
  - **Where functionality already exists**
  - **Which duplicate other triggers**
- **The excessive use of triggers can result in complex interdependencies, which may be difficult to maintain in large applications.**

Database Trigger: Example

Application

```
SQL> INSERT INTO EMP
  2  . . .;
```

EMP table

EMPNO	ENAME	JOB	SAL
7838	KING	PRESIDENT	5000
7698	BLAKE	MANAGER	2850
7369	SMITH	CLERK	800
7788	SCOTT	ANALYST	3000

CHECK_SAL trigger

Creating Triggers

- Trigger timing
  - For table: BEFORE, AFTER
  - For view: INSTEAD OF
- Triggering event: INSERT, UPDATE, or DELETE
- Table name: On table or view
- Trigger type: Row or statement
- When clause: Restricting condition
- Trigger body: PL/SQL block

DML Trigger Components

- **Trigger timing: When should the trigger fire?**
  - **BEFORE: Execute the trigger body before the triggering DML event on a table.**
  - **AFTER: Execute the trigger body after the triggering DML event on a table.**
  - **INSTEAD OF: Execute the trigger body instead of the the triggering statement. Used for VIEWS that are not otherwise modifiable.**

DML Trigger Components

- **Triggering user event:**

  What DML statement will cause the trigger to execute?
  - **INSERT**
  - **UPDATE**
  - **DELETE**

DML Trigger Components

- **Trigger type:**
- **How many times should the trigger body execute when the triggering event takes place?**
  - **Statement: The trigger body executes once for the triggering event. This is the default.**
  - **Row: The trigger body executes once for each row affected by the triggering event.**
- **Trigger body:**

  What action should the trigger perform?
  - **The trigger body is a PL/SQL block or a call to a procedure.**

Firing Sequence

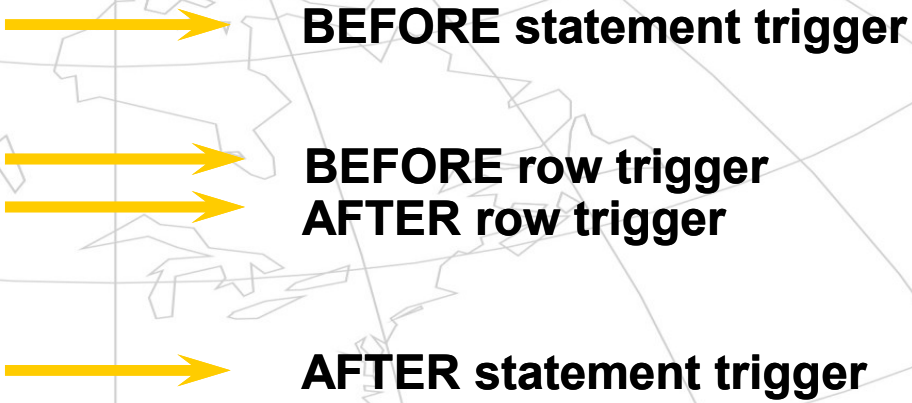
Firing sequence of a trigger on a table, when a single row is manipulated:

- **DML Statement**

```
SQL> INSERT INTO dept (deptno, dname, loc)
  2  VALUES (50, 'EDUCATION', 'NEW YORK');
```

- **Triggering Action**

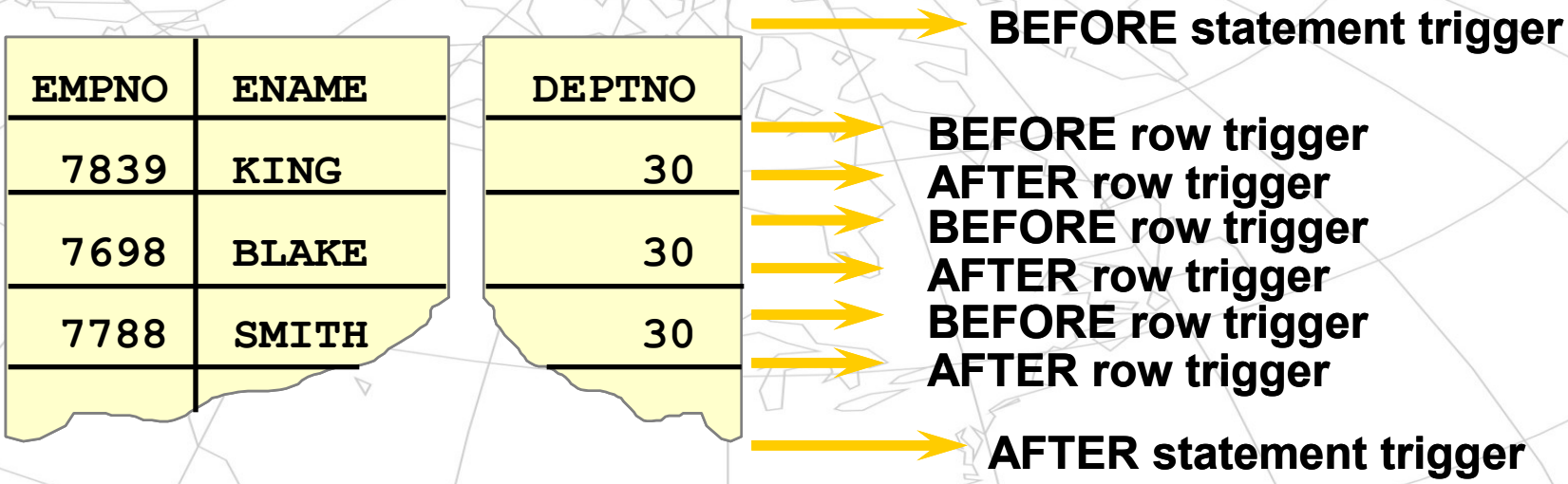
DEPTNO	DNAME	LOC
10	ACCOUNTING	NEW YORK
20	RESEARCH	DALLAS
30	SALES	CHICAGO
40	OPERATIONS	BOSTON



Firing Sequence

Firing sequence of a trigger on a table, when many rows are manipulated:

```
SQL> UPDATE emp
  2  SET sal = sal * 1.1
  3  WHERE deptno = 30;
```



Syntax for Creating Statement Triggers

```
CREATE [OR REPLACE] TRIGGER trigger_name
   timing
   event1 [OR event2 OR event3]
   ON table_name
trigger_body
```

Creating Statement Triggers Using SQL*Plus

```
CREATE OR REPLACE TRIGGER secure_emp
BEFORE INSERT ON employees
BEGIN
    IF (TO_CHAR(SYSDATE,'DY') IN ('SAT','SUN')) OR
    (TO_CHAR(SYSDATE,'HH24:MI')
    NOT BETWEEN '08:00' AND '18:00')
    THEN
    RAISE_APPLICATION_ERROR (-20500,'You may
    insert into EMPLOYEES table only
    during business hours.');
    END IF;
END;
```

Testing SECURE_EMP

```
INSERT INTO employees (employee_id, last_name,
first_name, email, hire_date,
job_id, salary, department_id)
VALUES (300, 'Smith', 'Rob', 'RSMITH', SYSDATE,
'IT_PROG', 4500, 60);INSERT INTO emp (empno,
   ename, deptno);
```

```
INSERT INTO employees (employee_id, last_name, first_name, email,
        *
ERROR at line 1:
ORA-20500: You may insert into EMPLOYEES table only during business hours.
ORA-06512: at "PLSQL.SECURE_EMP", line 4
ORA-04088: error during execution of trigger 'PLSQL.SECURE_EMP'
```

Using Conditional Predicates

```
CREATE OR REPLACE TRIGGER secure_emp
BEFORE INSERT OR UPDATE OR DELETE ON employees
BEGIN
      IF (TO_CHAR (SYSDATE,'DY') IN ('SAT','SUN')) OR
      (TO_CHAR (SYSDATE, 'HH24') NOT BETWEEN '08' AND '18')
      THEN
            IF DELETING THEN
                  RAISE_APPLICATION_ERROR (-20502,'You may delete from
                  EMPLOYEES table only during business hours.');
            ELSIF INSERTING THEN
                  RAISE_APPLICATION_ERROR (-20500,'You may insert into
                  EMPLOYEES table only during business hours.');
            ELSIF UPDATING ('SALARY') THEN
                  RAISE_APPLICATION_ERROR (-20503,'You may update
                  SALARY only during business hours.');
            ELSE
                  RAISE_APPLICATION_ERROR (-20504,'You may update
                  EMPLOYEES table only during normal hours.');
            END IF;
      END IF;
END;
```

Creating a Row Trigger

```
CREATE [OR REPLACE] TRIGGER trigger_name
  timing
    event1 [OR event2 OR event3]
      ON table_name
  [REFERENCING OLD AS old | NEW AS new]
FOR EACH ROW
  [WHEN condition]
trigger_body
```

Creating Row Triggers Using SQL*Plus

```
CREATE OR REPLACE TRIGGER restrict_salary
BEFORE INSERT OR UPDATE OF salary ON employees
FOR EACH ROW
BEGIN
    IF NOT (:NEW.job_id IN ('AD_PRES', 'AD_VP'))
    AND :NEW.salary > 15000
    THEN
        RAISE_APPLICATION_ERROR (-20202,'Employee
        cannot earn this amount');
    END IF;
END;
/
```

Using OLD and NEW Qualifiers

```
CREATE OR REPLACE TRIGGER audit_emp_values
AFTER DELETE OR INSERT OR UPDATE ON employees
FOR EACH ROW
BEGIN
INSERT INTO audit_emp_table (user_name, timestamp,
id, old_last_name, new_last_name, old_title,
new_title, old_salary, new_salary)
VALUES (USER, SYSDATE, :OLD.employee_id,
:OLD.last_name, :NEW.last_name, :OLD.job_id,
:NEW.job_id, :OLD.salary, :NEW.salary );
END;
/
```

Example Using Audit_Emp_Table

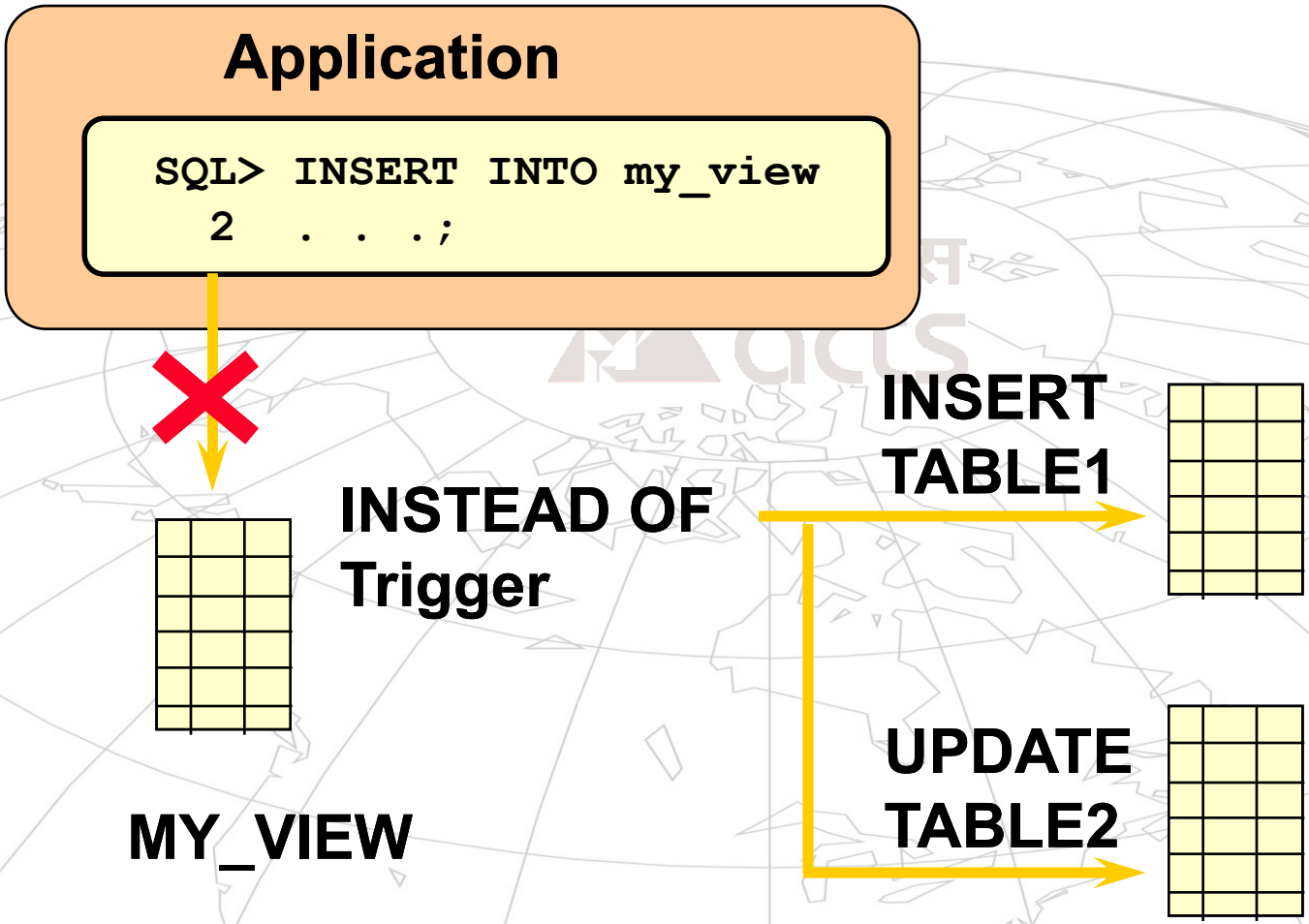
- **UPDATE employees**
  SET salary = 2000, last_name = 'Smith'
  WHERE employee_id = 999;

  SELECT *
  FROM audit_emp_table;

Restricting a Row Trigger

```
CREATE OR REPLACE TRIGGER derive_commission_pct
BEFORE INSERT OR UPDATE OF salary ON employees
FOR EACH ROW
WHEN (NEW.job_id = 'SA_REP')
BEGIN
    IF INSERTING
    THEN :NEW.commission_pct := 0;
    ELSIF :OLD.commission_pct IS NULL
    THEN :NEW.commission_pct := 0;
    ELSE
    :NEW.commission_pct := :OLD.commission_pct + 0.05;
    END IF;
END;
```

An INSTEAD OF Trigger



Creating an INSTEAD OF Trigger

```
CREATE [OR REPLACE] TRIGGER trigger_name
  INSTEAD OF
    event1 [OR event2 OR event3]
      ON view_name
   [REFERENCING OLD AS old | NEW AS new]
[FOR EACH ROW]
trigger_body
```

Creating an INSTEAD OF Trigger

```
INSERT INTO EMP_DETAILS (EMPNO, ENAME, SAL, DEPTNO)
VALUES (9001,'ABBOTT',1000,10)
```

INSTEAD OF INSERT into EMP_DETAILS

EMPNO	ENAME	SAL	DEPTNO	DNAME	TOT_DEPT_SAL
7836	KING	5000	10	NEW YORK	8750
7782	CLARK	2450	10	NEW YORK	8750
7934	MILLER	1300	10	NEW YORK	8750
7566	JONES	2975	20	DALLAS	10875

INSERT into EMPLOYEES

EMPNO	ENAME	SAL
7939	KING	5000
7698	BLAKE	2850
7782	CLARK	2450
9001	ABBOTT	1000

UPDATE DEPARTMENTS

DEPTNO	DNAME	TOT_DEPT_SAL
10	ACCOUNTING	9750
20	RESEARCH	10875
30	SALES	9400

Differentiating Between Triggers and Procedures

Triggers	Procedure
Use CREATE TRIGGER	Use CREATE PROCEDURE
Data dictionary contains source and p-code	Data dictionary contains source and p-code
Implicitly invoked	Explicitly invoked
COMMIT, SAVEPOINT, ROLLBACK not allowed	COMMIT, SAVEPOINT, ROLLBACK allowed

Managing Triggers

Disable or re-enable a database trigger:

```
ALTER TRIGGER trigger_name  DISABLE | ENABLE
```

```
ALTER TABLE table_name
DISABLE | ENABLE  ALL TRIGGERS
```

Recompile a trigger for a table:

```
ALTER TRIGGER trigger_name COMPILE
```

DROP TRIGGER Syntax

To remove a trigger from the database, use the DROP TRIGGER syntax:

```
DROP TRIGGER trigger_name
```

```
SQL> DROP TRIGGER secure_emp;
Trigger dropped
```

Trigger Execution Model and Constraint Checking

1. Execute all BEFORE STATEMENT triggers
2. Loop for each row affected
   a. Execute all BEFORE ROW triggers
   b. Execute the DML statement and perform integrity constraint checking
   c. Execute all AFTER ROW triggers
3. Execute all AFTER STATEMENT triggers

Summary

- Database triggers are used to perform related operations on events
- Database triggers can be written for DML, DDL and database events
- The instead of trigger can be written on views
- Trigger Execution sequence
- Managing Triggers

Thank You !